

Grundlagen der Informatik III

Wintersemester 2013/2014

Prof. Dr.-Ing. Michael Goesele,
Simon Fuhrmann, Fabian Langguth



TECHNISCHE
UNIVERSITÄT
DARMSTADT

2. Aufgabenblatt

24.10.2013

Die Aufgaben der Präsenzübung sollen in den Übungsgruppen bearbeitet werden. Die Hausaufgaben sind **vom 31.10. bis 06.11.** bei Ihrem jeweiligen Tutor zu Ihrer Präsenzübungszeit in handschriftlicher Form abzugeben. Die Abgaben sollen alle zum Nachvollziehen der Lösungen notwendigen Zwischenschritte, sowie den Namen des Tutors und die Übungsgruppe enthalten. Die Abgaben erfolgen in Teams mit drei Personen, wobei alle Mitglieder des Teams derselben Übungsgruppe angehören müssen. Jedes der Mitglieder muss eine eigene, handschriftliche Lösung abgeben und die anderen Mitglieder sind auf den Lösungen deutlich anzugeben.

Aufgabe 1: Registersatz/Programmiermodell

Beschreiben Sie kurz den Registersatz, der Ihnen bei der Intel IA32-Architektur zur Verfügung steht.

Aufgabe 2: Maschinensprache vs. Hochsprache

- (a) Wann ist es sinnvoll, Programme direkt in Assembler (statt in C oder JAVA) zu schreiben?
- (b) Nennen Sie einige Vorteile einer Hochsprache wie C gegenüber Assembler.

Aufgabe 3: Grundrechenarten

In dieser Aufgabe sollen die nachfolgenden Formeln in äquivalenten Assembler-Code umgewandelt werden. Dabei steht der Operator $:=$ für eine Zuweisung, die Variablen a, b, c sind in den Registern `%eax`, `%ebx`, `%ecx` abgelegt. Eine Überlaufbehandlung müssen Sie nicht durchführen. Beachten Sie jedoch, dass es sich bei allen Werten und Variablen um **long-Werte** handelt. Als Assembler-Syntax sollten Sie das AT&T-Format, das auch in der Vorlesung verwendet wird, benutzen.

- (a) $a := 23$
- (b) $a := a + b$
- (c) $c := a - b$
- (d) $a := 7 \cdot b$
- (e) $a := b^4 + (c + 1) \cdot b^2 + 1$


Aufgabe 4: Kontrollstrukturen

Wandeln Sie die folgenden Codebeispiele in IA32-Assembler um. Die Variablen a, b, c sind in den Registern `%eax`, `%ebx`, `%ecx` abgelegt.

- (a)

```
if (a < b) c++;
```
- (b)

```
c = 0;
while (a > 0)
{
    a -= b;
```



```
    c++;  
}
```

(c) `a = (b < c/2) ? b : c;`

Diese Anweisung ist eine Kurzschreibweise für `if (b < c/2) a = b; else a = c;`

(d)

```
a = 0;  
for (b = 0; b < 1024; b++)  
{  
    a += b;  
}
```

Hausaufgabe 1: Reverse-Engineering (4 Punkte)

Im Folgenden sind zwei Assembler-Programme gegeben.

<pre>.data a: .long 23 b: .long 5 .text movl a, %eax movl b, %ebx movl %ebx, %ecx shll \$1, %ebx subl %ebx, %eax shll \$1, %eax addl %eax, %ecx incl %eax</pre>	<pre>.data a: .long 23 b: .long 5 .text movl a, %eax movl b, %ebx addl %ebx, %ebx subl %ebx, %eax addl %eax, %eax addl %eax, b addl \$1, %eax movl b, %ecx</pre>
--	---

- (a) (2 Pkte.) Geben Sie jeweils die Registerinhalte nach dem Durchlauf aller Befehle an.
- (b) (2 Pkte.) Vergleichen Sie die beiden Assembler-Programme bezüglich Ihrer Ausführungszeit (in Takten), indem Sie für die folgenden Befehle die angegebenen Taktzahlen annehmen:

```
movl Quelle, Ziel
movl Reg, Reg      : 2 Takte
movl Reg, Speicher : 9 Takte
movl Speicher, Reg : 8 Takte
movl Wert, Reg     : 4 Takte
movl Wert, Speicher : 10 Takte

addl/subl Quelle, Ziel
addl/subl Reg, Reg      : 3 Takte
addl/subl Reg, Speicher : 16 Takte
addl/subl Speicher, Reg : 9 Takte
addl/subl Wert, Reg     : 4 Takte

incl Quelle
incl Reg      : 3 Takte
incl Speicher : 15 Takte

shll n, Ziel
shll $1, Reg      : 2 Takte
shll %cl, Reg     : 8 + 4 * n Takte
shll $1, Speicher : 15 Takte
shll %cl, Speicher : 20 + 4 * n Takte
```

Hausaufgabe 2: Primzahlen in Assembler (6 Punkte)

Schreiben Sie ein Assemblerprogramm, dass zu einer gegebenen Zahl die nächst größere Primzahl sucht und im Register %eax ablegt. Sie können folgendes Codegerüst verwenden, welches den Wert im Register am Ende auf die Standardausgabe schreibt. Die Eingabe des Programms ist in der Speicherstelle a vorgegeben. Wie schon in der Vorlesung vorgeführt können sie den Code auf den Rechnern der RBG mit folgenden Befehlen kompilieren und ausführen (angenommen Sie haben die Datei 'prim.S' erstellt):

```
>> gcc -m32 -o prim prim.S
>> ./prim
```

Ihr Code muss auf den Rechnern im RBG-Pool lauffähig sein.

Sie müssen den Code sowohl bei Ihrem Tutor in der Übung abgeben (gedruckter Code ist erlaubt), als auch im Moodle unter der entsprechenden Abgabe hochladen.

Codegerüst:

```
.data
a: .long 23
intout: .string "Naechste Primzahl:_%d\n"

.text
.globl main
main:

# ... Hier Ihren Code einfuegen ...

# Wert im %eax ausgeben
pushl %eax
pushl $intout
call printf
popl %eax
popl %eax
# Exit
movl $1, %eax
int $0x80
```

Als Anhaltspunkt für Ihre Implementierung können Sie folgenden Pseudo-Code verwenden:

```
a = 23;
next = a + 1;
while(true) {
    i = 2;
    while(i < next) {
        if(next % i == 0) {           // % entspricht Division mit Rest (modulo)
            next = next + 1
            break;
        } else {
            i = i + 1
        }
    }
    if(i == next) break;
}
print next;
```

Hinweis: Wenn Sie die Division mit dem Befehl `divl` ausführen wird Ihnen der Rest, der bei der Division entsteht, im Register `%edx` zurückgegeben.